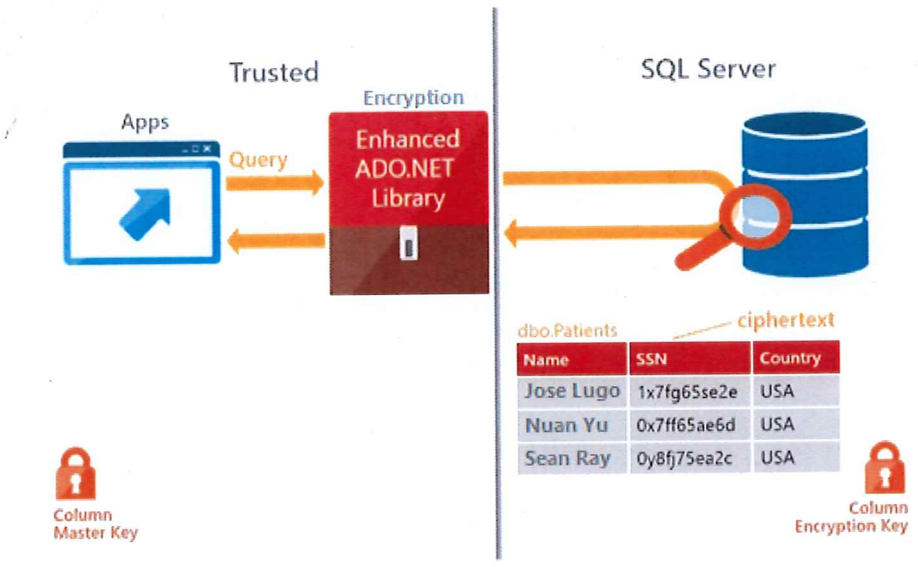
SQL Server 2016 Always Encrypted

SQL Server 2016 provides you a new way to encrypt data in the table columns called **Always Encrypted**. With Always Encrypted, data is encrypted at the application layer via ADO.NET. This means you can encrypt your confidential or customer sensitive data with your .NET application prior to the data being sent across the network to SQL Server. In this article I will explain setting up a table that stores always encrypted data.

Always Encrypted Architecture

The architecture for Always Encrypted has the application performing the column level data encryption prior to the confidential data from being sent to the SQL Server. The actual encryption is done by the ADO.NET drivers on an application, or client machine. When a .NET application sends plain text data to ADO.NET it is encrypted prior to sending it to SQL Server. The only change to store encrypted data that the application needs to make is to change the connection string to indicate column encryption is enabled. When column encryption is enabled ADO.NET will encrypt Always Encrypted columns data prior to sending to the SQL Server, and will decrypted Always Encrypted columns data when they are read from the SQL Server. The following diagram shows Always Encrypted Architecture.



Name	SSN	Country
Jose Lugo	1x7fg65se2e	USA
Nuan Yu	0x7ff65ae6d	USA
Sean Ray	0y8fj75ea2c	USA

Always Encrypted Architecture

In this diagram you can see two different kinds of keys: Column Master Key, and Column Encryption Key.

The Column Master Key is stored on an application machine, in an external key store. This key is used to protect the Column Encryption key. By placing this key on the application machine SQL Server doesn't have access to the column master key directly. Therefore SQL Server by itself will not be able decrypt the Always Encrypted data.

The other key, the Column Encryption Key, is stored on SQL Server. This key is used to encrypt/decrypt the Always Encrypted columns. Once ADO.NET has decrypted the Column Encryption Key, using the Column Master Key it can use the decrypt Column Encryption Key to decrypt/encrypt Always Encrypted columns.

Components Required to Implement Always Encrypted

We need the following components to store Always Encrypted column in a SQL Server table:

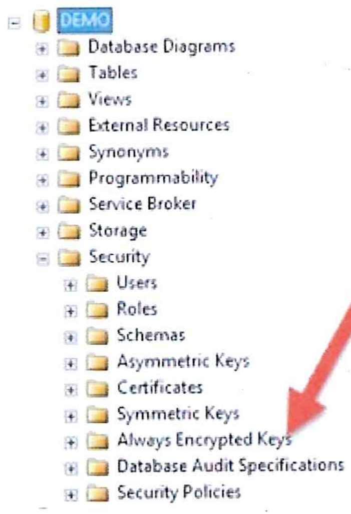
- An application that uses .NET 4.6 framework
- A SQL Server 2016 instance
- A certificate store to support the Column Master Key
- A Column Master Key
- A Column Encryption Key
- A table with Always Encrypted columns

.NET Framework 4.6 must be installed in the machine hosting your client application. .NET Framework 4.6 is available with SQL Server 2016 Community Technology Preview 3 (CTP 3.0) and is installed with SQL Server Management Studio. If your client application runs on a machine that does not contain SQL Server 2016 Community Technology Preview 3 (CTP 3.0) Management Studio, you need to install at least .NET Framework 4.6 (for details, see .NET Framework 4.6).

Creating Column Master Key and Column Encryption Key

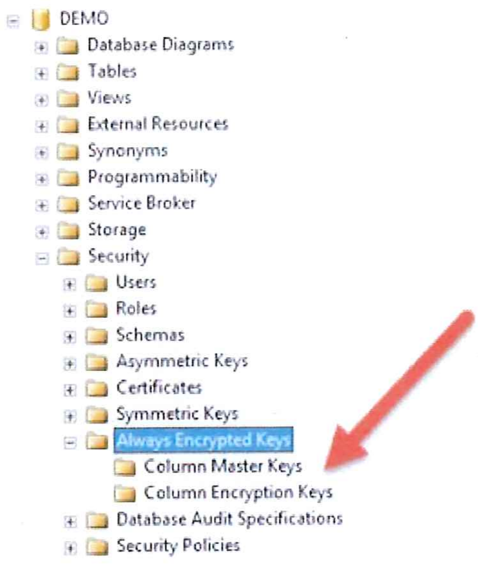
I have created a DEMO database in the SQL Server 2016 instance. I will use this database to store the table that will contain Always Encrypted Columns.

Next I will show how to create Column Master Key using SSMS. Expand the DEMO database in SSMS, and then expand the "Security" item. Under the "Security" you can see the "Always Encrypted Keys" item, as shown below.



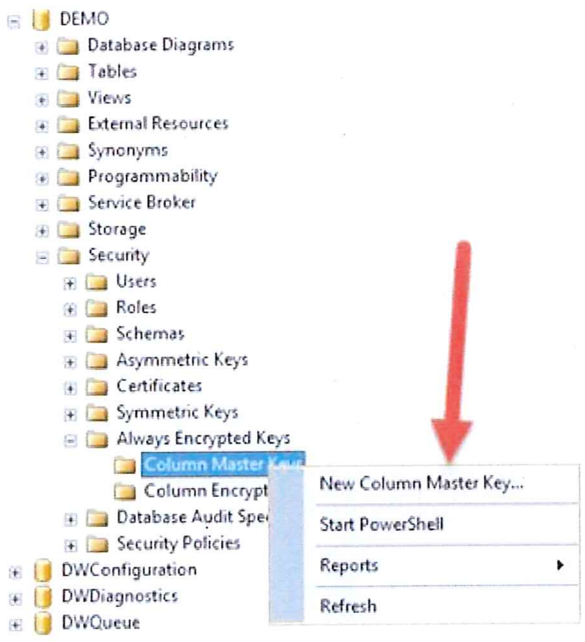
Always Encrypted Keys

Expand the "Always Encrypted Keys" item and you will find the Column Master Keys and Column Encryption Keys items as show below:



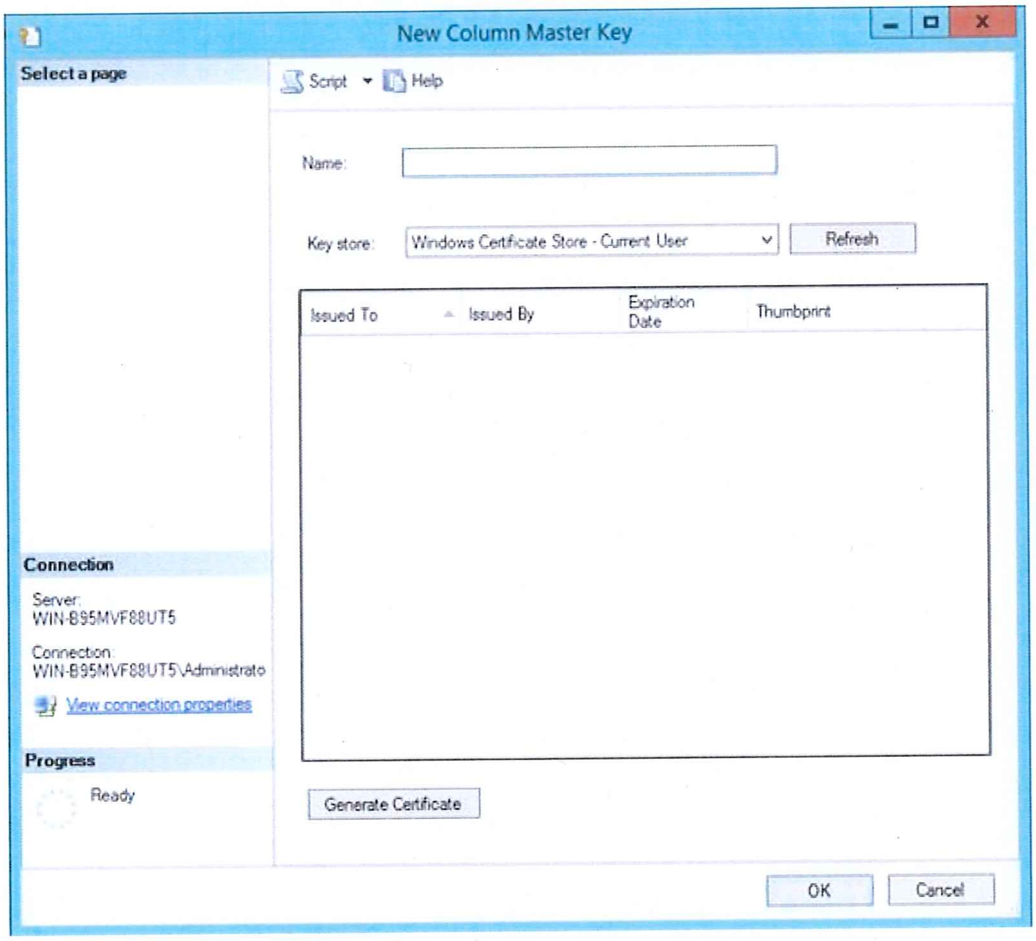
Column Master Keys and Column Encryption Keys

To create Column Master Key, right click on the "Column Master Keys" item in the object explorer and select "New Column Master Key…" item as show below:



New Column Master Key…

The following screen will be displayed.



Name:

Key store: Windows Certificate Store - Current User

Issued To	Issued By	Expiration Date	Thumbprint

Connection

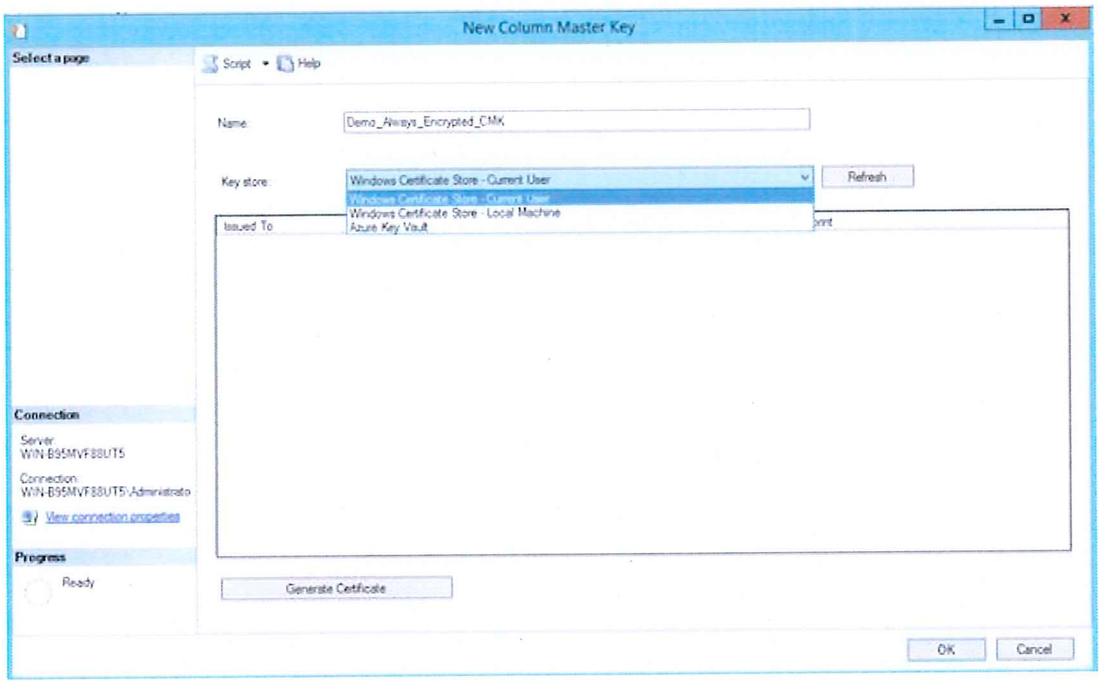
Server:
WIN-B95MVF88UT5

Connection:
WIN-B95MVF88UT5\Administrato

Progress

Ready

Enter the "Name" of the Column Master Key. Name it as "Demo_Always_Encrypted_CMK". For the "Key store", expand the drop down box to see the different key store options, see below:



New Column Master Key

Name: Demo_Always_Encrypted_CMK

Key store: Windows Certificate Store - Current User

- Windows Certificate Store - Current User
- Windows Certificate Store - Local Machine
- Azure Key Vault

Connection

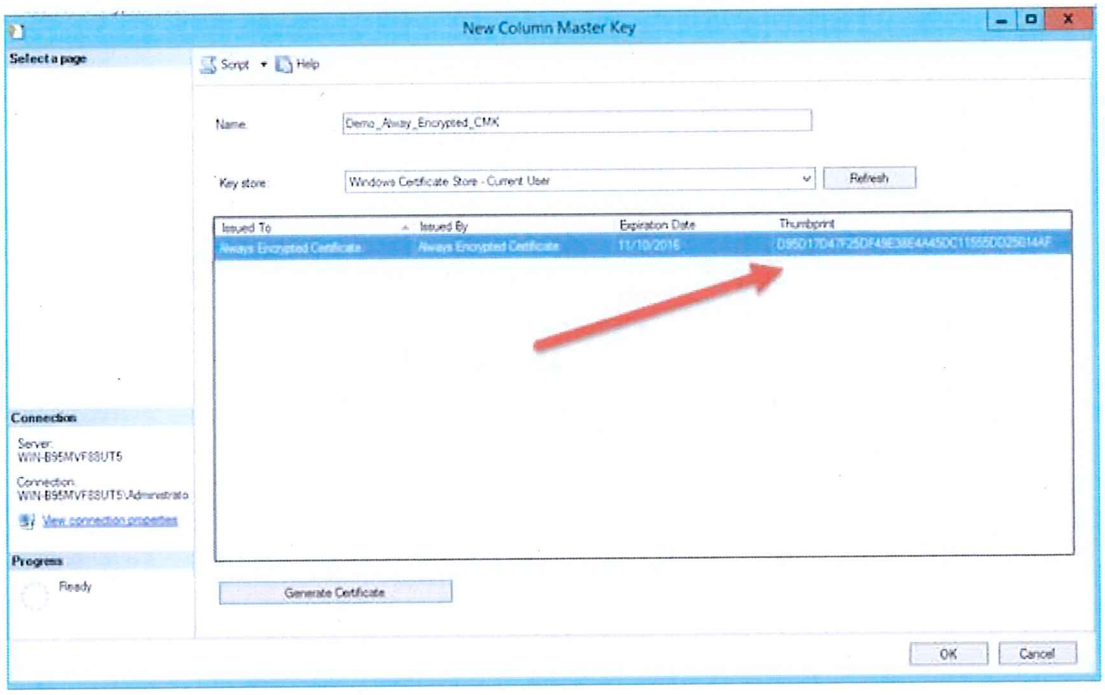
Server: WIN-B95MVF88UT5

Connection: WIN-B95MVF88UT5\Administrato

Progress: Ready

You have a choice of three different key store locations: Window Certificate Store – Current User, Window Certificate Store – Local Machine, Azure Key Vault. For our initial testing select the "Window Certificate Store – Current User", and then click on the "Generate Certificate" button.

The following window will be displayed:



New Column Master Key

Name: Demo_Alway_Encrypted_CMK

Key store: Windows Certificate Store - Current User

Issued To	Issued By	Expiration Date	Thumbprint
Always Encrypted Certificate	Always Encrypted Certificate	11/10/2016	D95D17D47F25DF49E38E4A45DC11555DD25014AF

Connection

Server: WIN-B95MVF88UT5

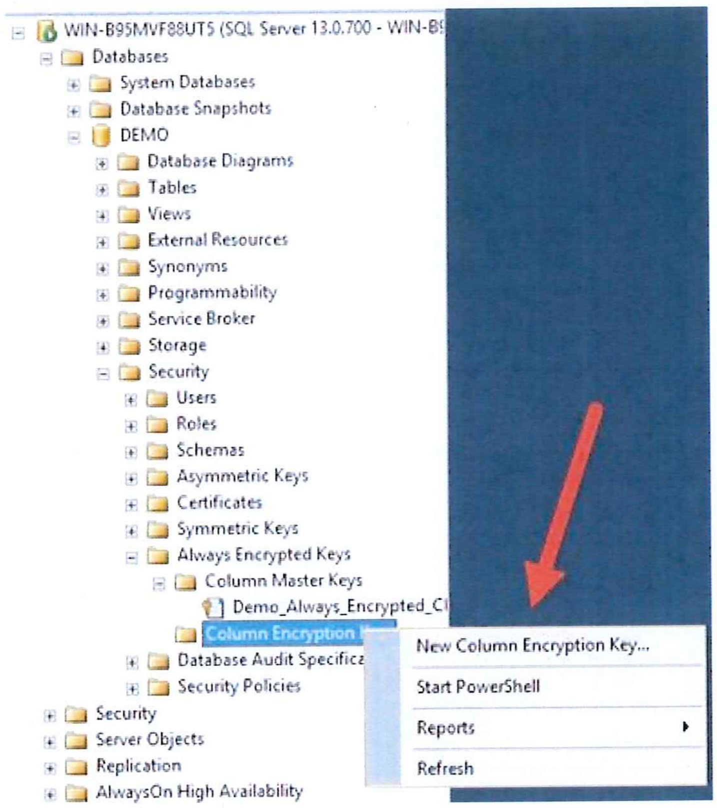
Connection: WIN-B95MVF88UT5\Administrato

Progress: Ready

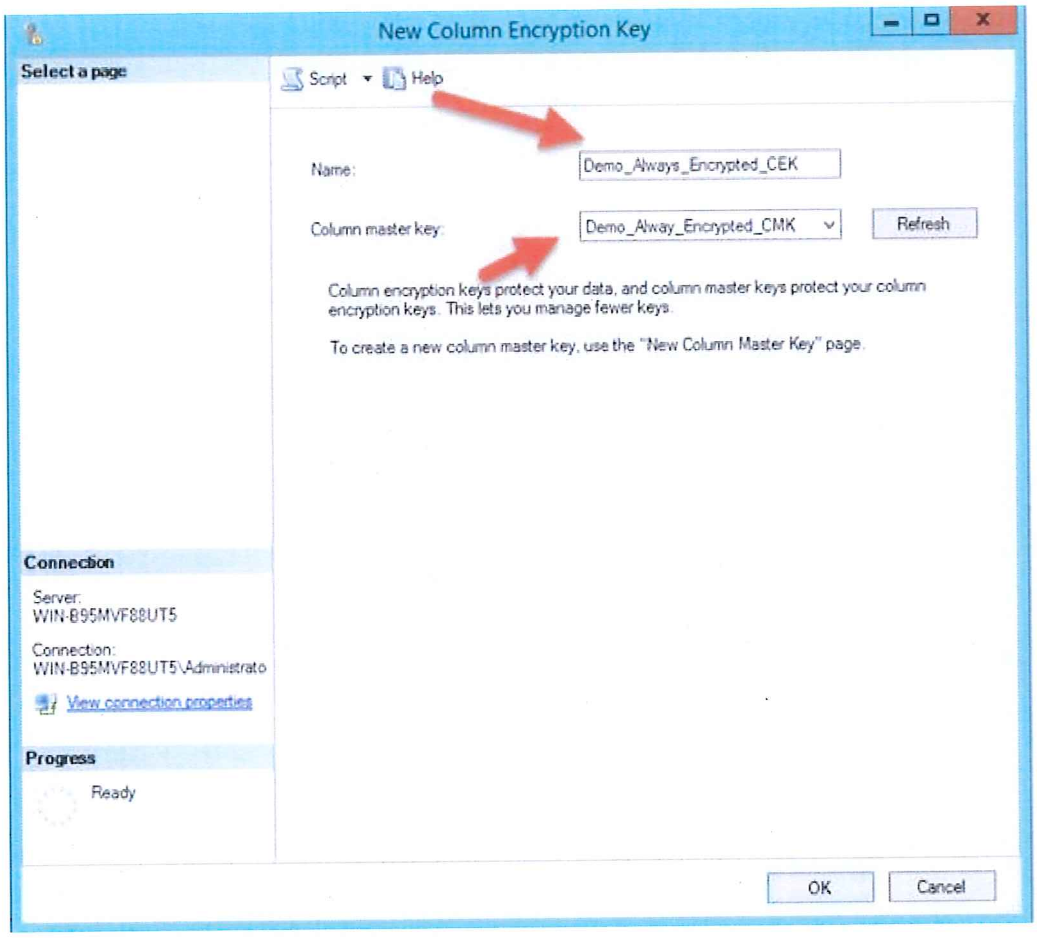
Here you can see that a new certificate was created, and there is a Thumbprint associated with it.

Now the “Demo_Always_Encrypted_CMK” column master key has been created.

Now we have to create a column encryption key. To create Column Encryption Key, right click on the “Column Encryption Keys” item in the object explorer and select “New Column Encryption Key…” item as show below:



The following screen will be displayed.



Name: Demo_Always_Encrypted_CEK

Column master key: Demo_Alway_Encrypted_CMK

Column encryption keys protect your data, and column master keys protect your column encryption keys. This lets you manage fewer keys.

To create a new column master key, use the "New Column Master Key" page.

Connection

Server:
WIN-B95MVF88UT5

Connection:
WIN-B95MVF88UT5\Administrato

View connection properties

Progress

Ready

Enter the name of the new column encryption key, which is “Demo_Always_Encrypted_CEK”. Select the “Column master key” “Demo_Always_Encrypted_CMK” from the drop down menu. Click the “OK” button to create the column encryption key.

Creating Always Encrypted Table

Now the column master key and column encrypted key have been created. We can create a table that will store always encrypted columns. Execute the following SQL code below create a table:

```
CREATE TABLE dbo.Demo_Always_Encrypted
(
  ID INT IDENTITY(1,1) PRIMARY KEY,
  LastName NVARCHAR(45),
  FirstName NVARCHAR(45),
  BirthDate DATE ENCRYPTED WITH
   (
     ENCRYPTION_TYPE = RANDOMIZED,
     ALGORITHM = 'AEAD_AES_256_CBC_HMAC_SHA_256',
     COLUMN_ENCRYPTION_KEY = Demo_Always_Encrypted_CEK
   ),
SSN NVARCHAR(10) COLLATE Latin1_General_BIN2
 ENCRYPTED WITH
 (
    ENCRYPTION_TYPE = DETERMINISTIC,
    ALGORITHM = 'AEAD_AES_256_CBC_HMAC_SHA_256',
    COLUMN_ENCRYPTION_KEY = Demo_Always_Encrypted_CEK
  ) );
```

In this code you can see that the BirthDate, and SSN are always encrypted columns. For the BirthDate column I created it with an encryption type of RANDOMIZED, whereas the second always encrypted column, SSN has an encryption type of DETERMINISTIC.

DETERMINISTIC encryption means that clear text of a given column value will always be encrypted to the same value. Whereas a RANDOMIZE encrypted column could get a different encrypted value each time the column value is encrypted. If you plan to encrypt a column for searching or joining then you will need to use an encryption type of DETERMINISTIC. You should use RANDOMIZE encryption type for columns used for displayed purposes only. DETERMINISTIC is less secure, because by using a brute force attack, eventually a hacker could determine the unencrypted value. Therefore using RANDOMIZE encrypted columns are more secure than using DETERMINISTIC encrypted columns.

Note: - When encrypting a string value you need set the Always Encrypted column to a BIN2 collation setting. By reviewing the code above you can see SSN is defined as nvarchar(10), with a collation of Latin1_General_BIN2.

Stored Procedure for Testing Always Encrypted

To insert data into the Always Encrypted table let's execute the following stored procedure:

```
CREATE PROCEDURE Insert_Always_Encrypted (
@LastName varchar(45),
@FirstName varchar(45),
@BirthDate date,
@SSN NVARCHAR(10))
AS
INSERT INTO dbo.Demo_Always_Encrypted
  (LastName, FirstName, BirthDate, SSN)
VALUES (@LastName,@FirstName,@BirthDate,@SSN);
```

As you can see, this stored procedure accepts parameter values for every column in the Demo_Always_Encrypted table. It then takes the passed parameter values and inserts them into the table.

Test Iteration #1 - Inserting Always Encrypted Data

In our first test of Always Encrypted let's call the Insert_Alway_Encrypted store procedure with the following code from within a query window within SQL Server Management Studio (SSMS):

```
EXEC Insert_Always_Encrypted @LastName = 'Larsen',
                     @FirstName = 'Gregory',
                     @BirthDate = '1950-01-01',
                     @SSN = '123-45-6789';
```

When you run this code we will get the following error:

```
Msg
206, Level 16, State 2, Procedure Insert_Always_Encrypted, Line 11
Operand type clash: varchar is incompatible with date encrypted with (encryption_type =
```

'DETERMINISTIC', encryption_algorithm_name = 'AEAD_AES_256_CBC_HMAC_SHA_256', column_encryption_key_name = 'Demo_Always_Encrypted_CEK', column_encryption_key_database_name = 'DEMO')

This error message is telling us that we sent a BirthDate column value in clear text, instead of it being an encrypted value. Remember the Always Encrypted architecture requires the encrypted column to be encrypted via ADO.NET, which didn't occur when we executed this code via SSMS.

Setting up for Test Iteration #2 - Inserting Always Encrypted Data

For the second test let's use a C# program to insert a record into the Demo_Always_Encrypted table. Once the record is inserted it will then be read back to verify the C# could read and decrypted the encrypted columns. The C# code for this test can be found at the bottom of this article.

In the C# code at the bottom of this article we will use the stored procedure "Insert_Always_Encrypted" to insert a row into the "Demo_Always_Encrypted" table. After the record is inserted, the code displays a message box that says "Inserted Demo Records..." Next the code reads the encrypted data by using a SELECT statement and lastly the code displays a message box showing the unencrypted data it read.

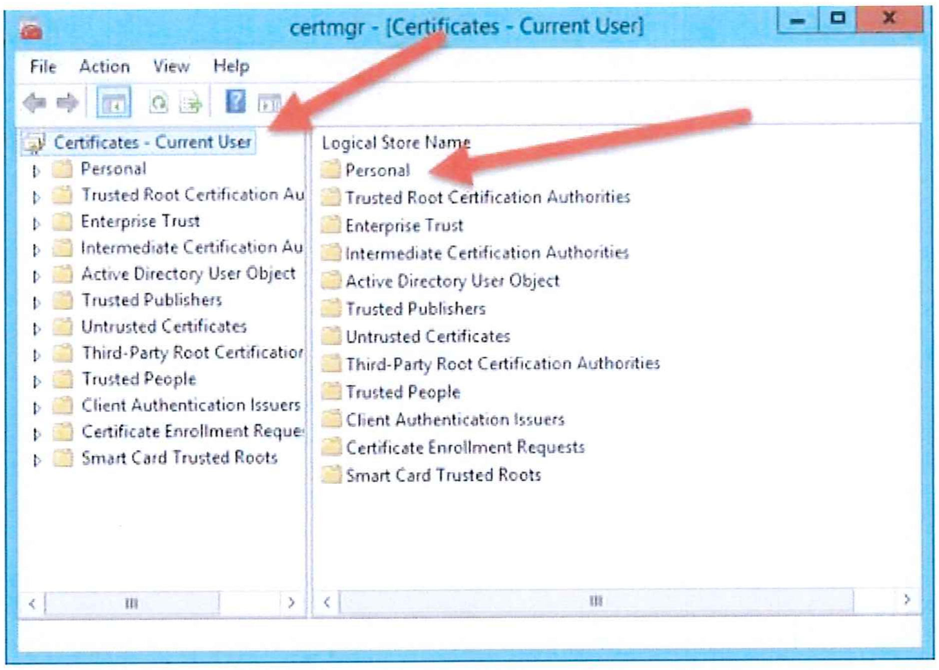
Before you use the C# code, you need to make sure your machine where you will be running the C# code has access to the Column Master Key value. We need to export the Column Master Key from the certificate store and import the exported certificate in the certificate store on your machine.

Exporting and Importing Column Master Key (CMK)

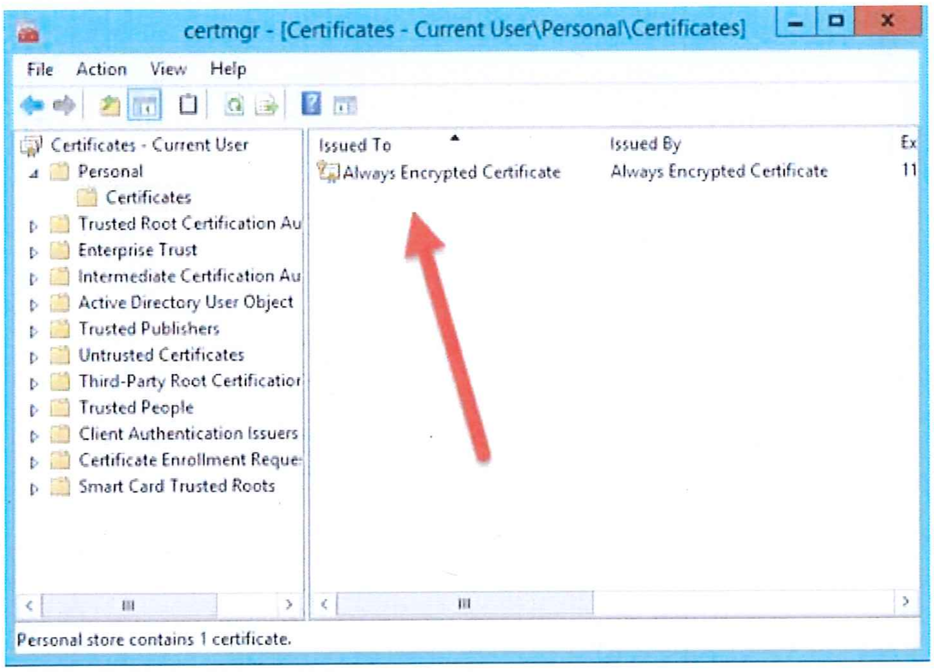
In order to export and import the CMK, I will use certmgr.exe; you can obtain it by downloading the Windows SDK (Software Development Kits) from the following location: http://go.microsoft.com/fwlink/p/?linkid=84091.

I installed the Windows SDK on both my VM machine and my laptop. If you already know how to use certmgr to import and export certificates then you can skip this section.

To start the export process I executed certmgr from my VM machine. The following window is displayed:


certmgr - [Certificates - Current User]
File
Action
View
Help
Certificates - Current User
Personal
Trusted Root Certification Au
Enterprise Trust
Intermediate Certification Au
Active Directory User Object
Trusted Publishers
Untrusted Certificates
Third-Party Root Certificatior
Trusted People
Client Authentication Issuers
Certificate Enrollment Reque
Smart Card Trusted Roots
Logical Store Name
Personal
Trusted Root Certification Authorities
Enterprise Trust
Intermediate Certification Authorities
Active Directory User Object
Trusted Publishers
Untrusted Certificates
Third-Party Root Certification Authorities
Trusted People
Client Authentication Issuers
Certificate Enrollment Requests
Smart Card Trusted Roots

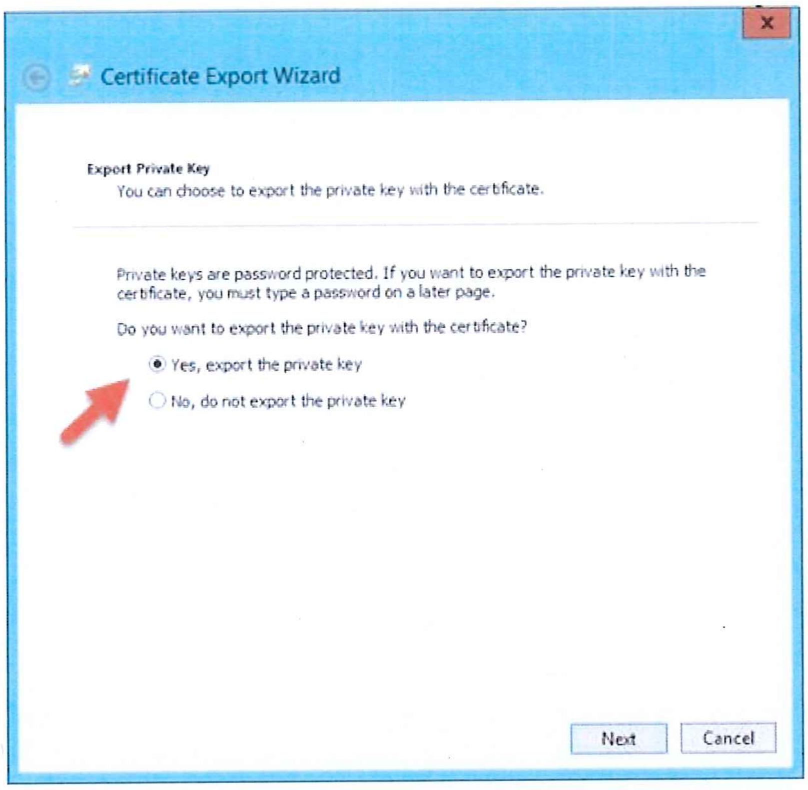
On this window you can see I'm browsing the "Current User" certificate store. Drill down and look at the certs under the "Personal" folder. You can find the following certificate:


certmgr - [Certificates - Current User\Personal\Certificates]
File
Action
View
Help
Certificates - Current User
Personal
Certificates
Trusted Root Certification Au
Enterprise Trust
Intermediate Certification Au
Active Directory User Object
Trusted Publishers
Untrusted Certificates
Third-Party Root Certificatior
Trusted People
Client Authentication Issuers
Certificate Enrollment Reque
Smart Card Trusted Roots

Issued To	Issued By	Ex
Always Encrypted Certificate	Always Encrypted Certificate	11

Personal store contains 1 certificate.

You can find the certificate that was created when we have created our Column Master Key. To export this certificate, right click on the certificate, and then click on the "All Tasks" item from the menu displayed and then finally click on the "Export…" task on the next window displayed. When you select the "Export…" task, a welcome window will display, click the "Next" button which brings up the "Certificate Export Wizard" as shown below:



Certificate Export Wizard

Export Private Key

You can choose to export the private key with the certificate.

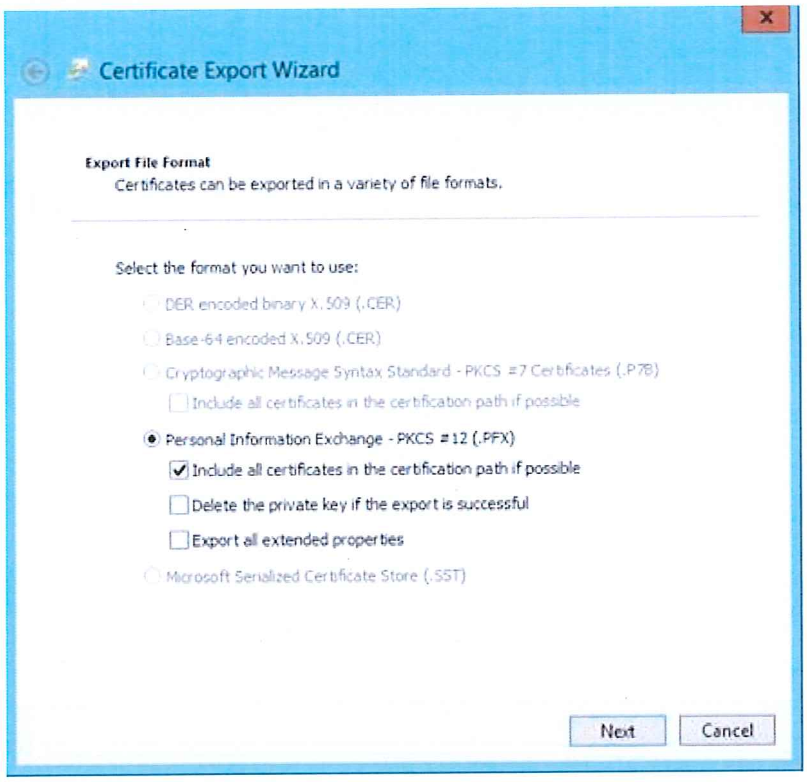
Private keys are password protected. If you want to export the private key with the certificate, you must type a password on a later page.

Do you want to export the private key with the certificate?

[✓ Yes, export the private key]

No, do not export the private key

On this screen select the "Yes, export the private key" radio button and then click on the "Next" screen. Upon doing this the following screen will be displayed:



Certificate Export Wizard

Export File Format

Certificates can be exported in a variety of file formats.

Select the format you want to use:

DER encoded binary X.509 (.CER)

Base-64 encoded X.509 (.CER)

Cryptographic Message Syntax Standard - PKCS #7 Certificates (.P7B)

Include all certificates in the certification path if possible

[✓ Personal Information Exchange - PKCS #12 (.PFX)]

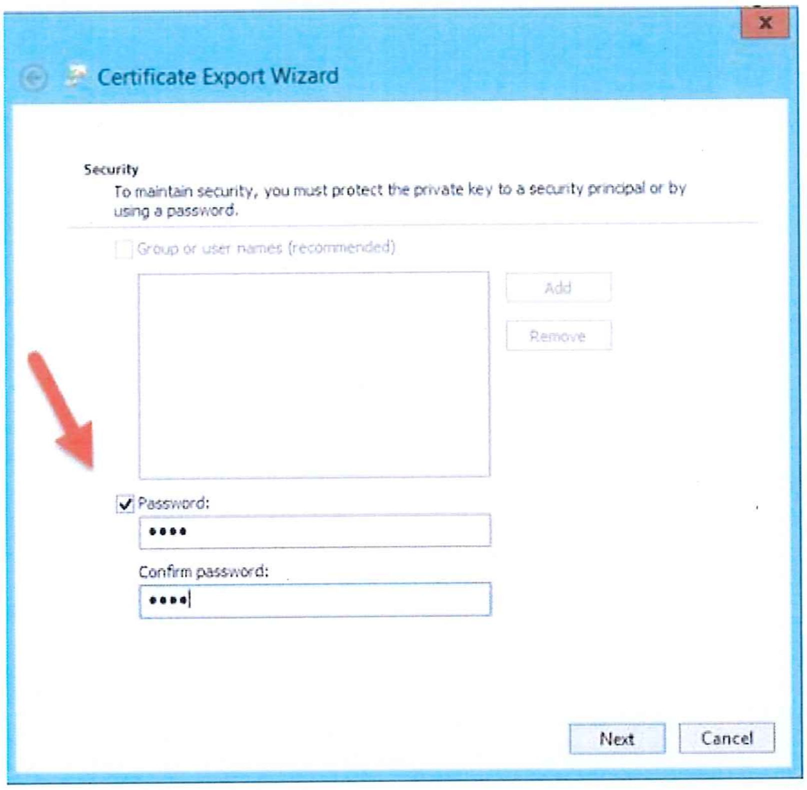
[✓ Include all certificates in the certification path if possible]

Delete the private key if the export is successful

Export all extended properties

Microsoft Serialized Certificate Store (.SST)

Here just take the defaults and click on the "Next" button. Doing this brings up the following window:



Certificate Export Wizard

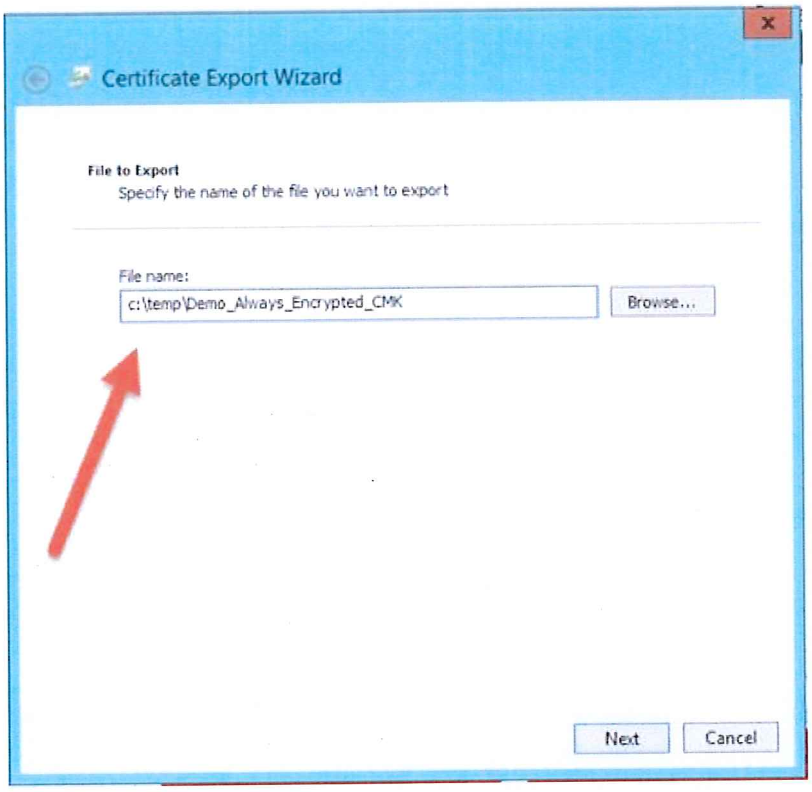
Security
To maintain security, you must protect the private key to a security principal or by using a password.

☐ Group or user names (recommended)

☑ Password:
••••

Confirm password:
••••

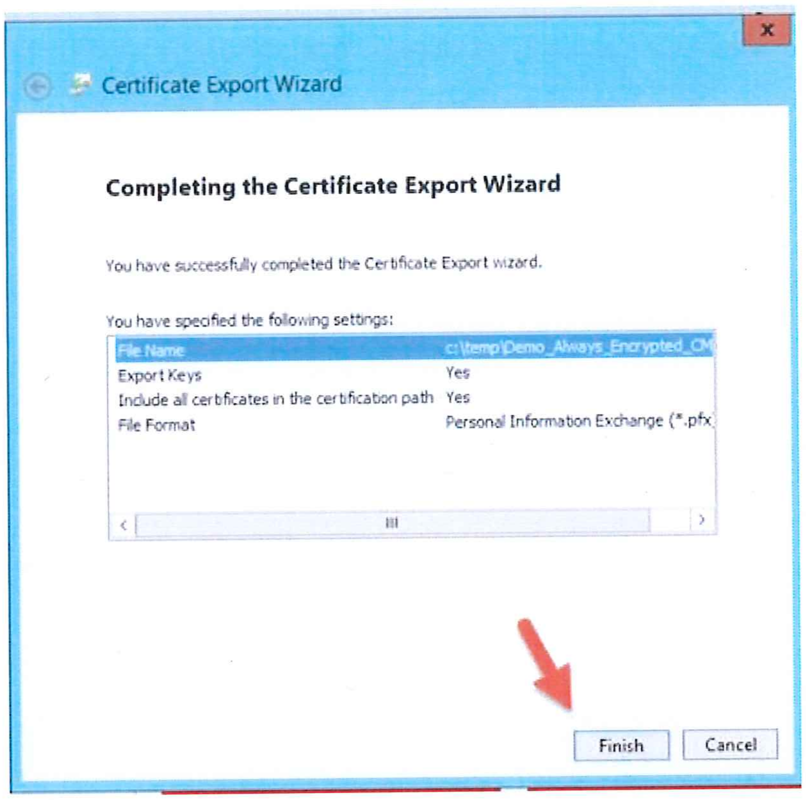
On this screen select the "Password:" checkbox, and then enter a password that will be associated with the exported certificate file. Once the password is entered, then click on the "Next" button, which brings up this window:


Certificate Export Wizard

File to Export
Specify the name of the file you want to export

File name:
c:\temp\Demo_Always_Encrypted_CMK

On this window enter a location and name for the exported certificate. As you can see we are going to export the certificate to a file named “Demo_Always_Encrypted_CMK”. Click on the “Next” button, the following window will be displayed:



Completing the Certificate Export Wizard

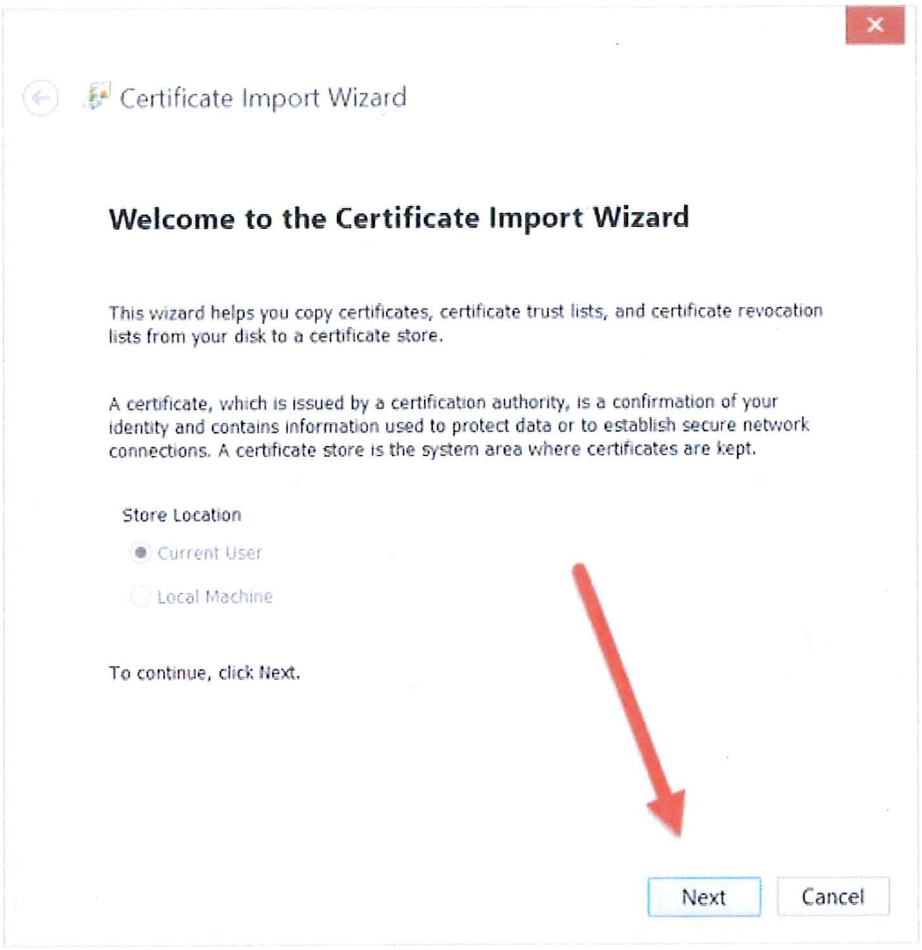
You have successfully completed the Certificate Export wizard.

You have specified the following settings:

File Name	c:\temp\Demo_Always_Encrypted_CM
Export Keys	Yes
Include all certificates in the certification path	Yes
File Format	Personal Information Exchange (*.pfx

Review the export settings and then click on the "Finish" button. A message box will be displayed saying that the certificate is exported.

To import the exported certificate, first copy the exported certificate from the VM machine to the C:\temp directory on to your laptop. Once the certificate file was copied, start the certmgr.msc on your laptop so you could import the exported cert file. In the certmgr interface, expand the Personal folder and then click on the "All Task:" item and finally select the "Import" option. When you do this the following screen will be displayed:


Certificate Import Wizard

Welcome to the Certificate Import Wizard

This wizard helps you copy certificates, certificate trust lists, and certificate revocation lists from your disk to a certificate store.

A certificate, which is issued by a certification authority, is a confirmation of your identity and contains information used to protect data or to establish secure network connections. A certificate store is the system area where certificates are kept.

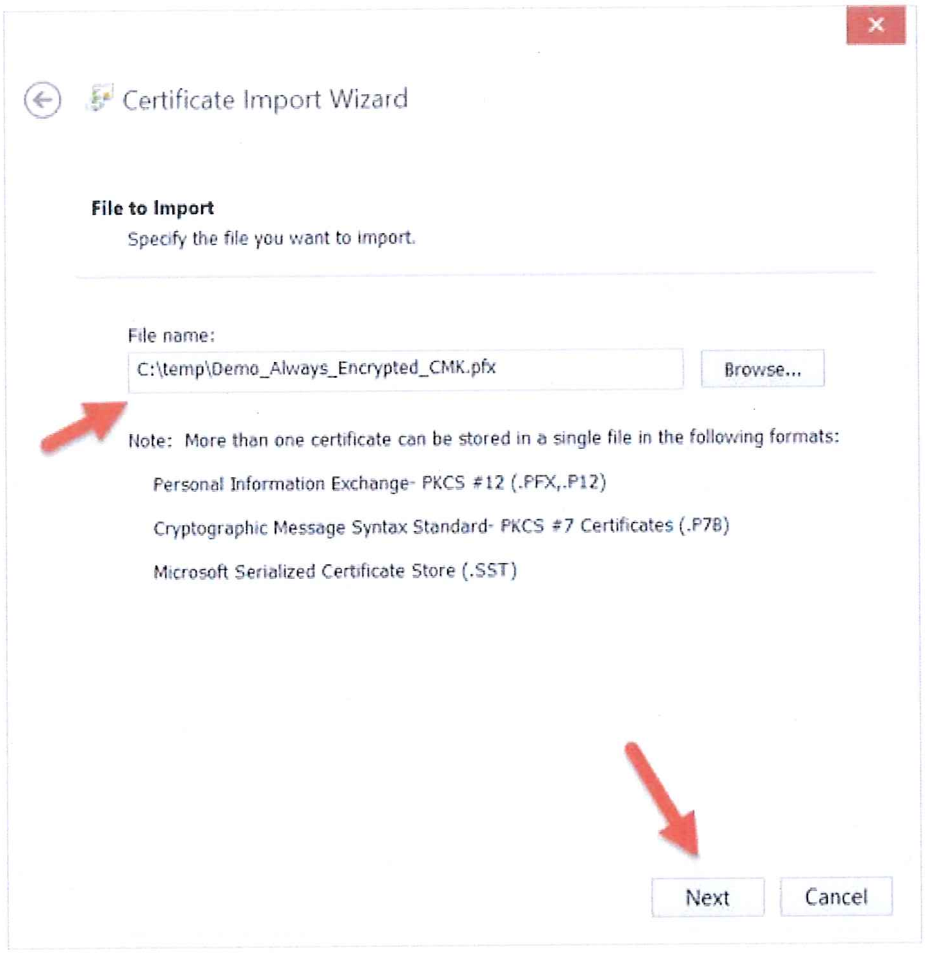
Store Location

(●) Current User

( ) Local Machine

To continue, click Next.

On this screen just click on the “Next” button, this displays the following window:



Certificate Import Wizard

File to Import

Specify the file you want to import.

File name:

C:\temp\Demo_Always_Encrypted_CMK.pfx

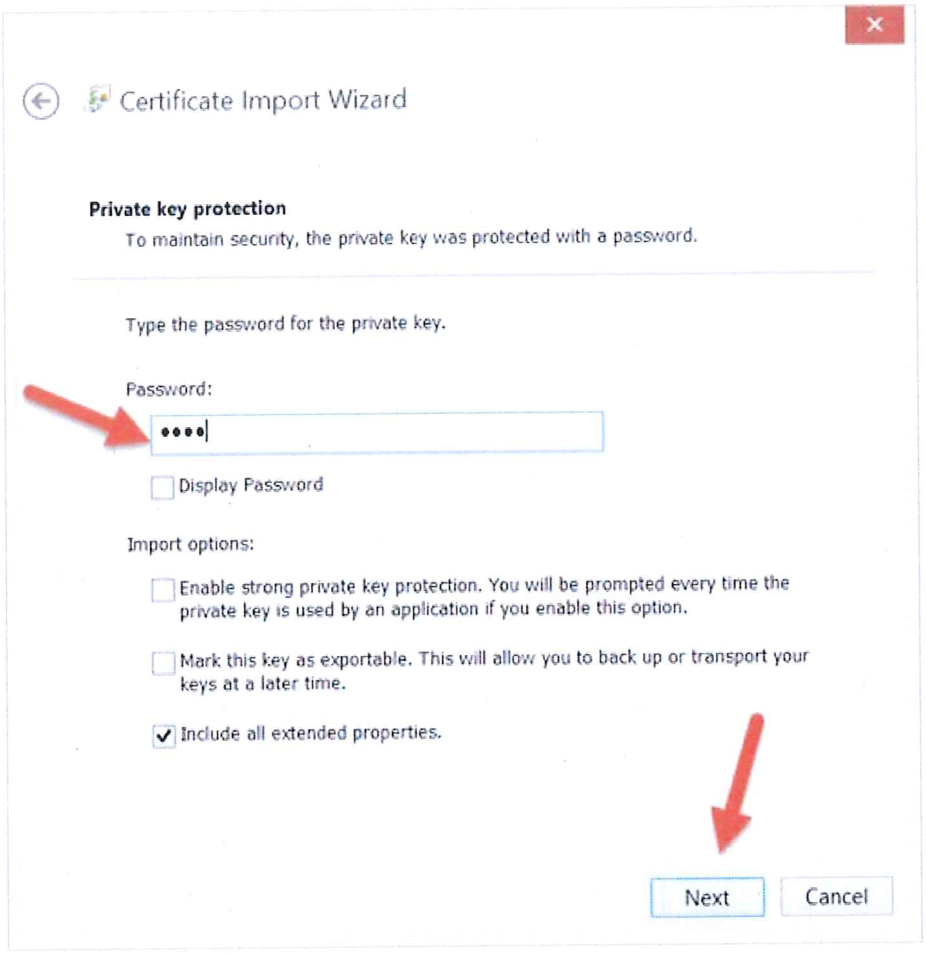
Note: More than one certificate can be stored in a single file in the following formats:

Personal Information Exchange- PKCS #12 (.PFX,.P12)

Cryptographic Message Syntax Standard- PKCS #7 Certificates (.P7B)

Microsoft Serialized Certificate Store (.SST)

On this screen browse for the CMK certificate that you copied to the C:\temp directory. After finding and selecting the certificate file, click the “Next” button. Upon doing that the following screen will be displayed:



Private key protection

To maintain security, the private key was protected with a password.

Type the password for the private key.

Password: ••••

☐ Display Password

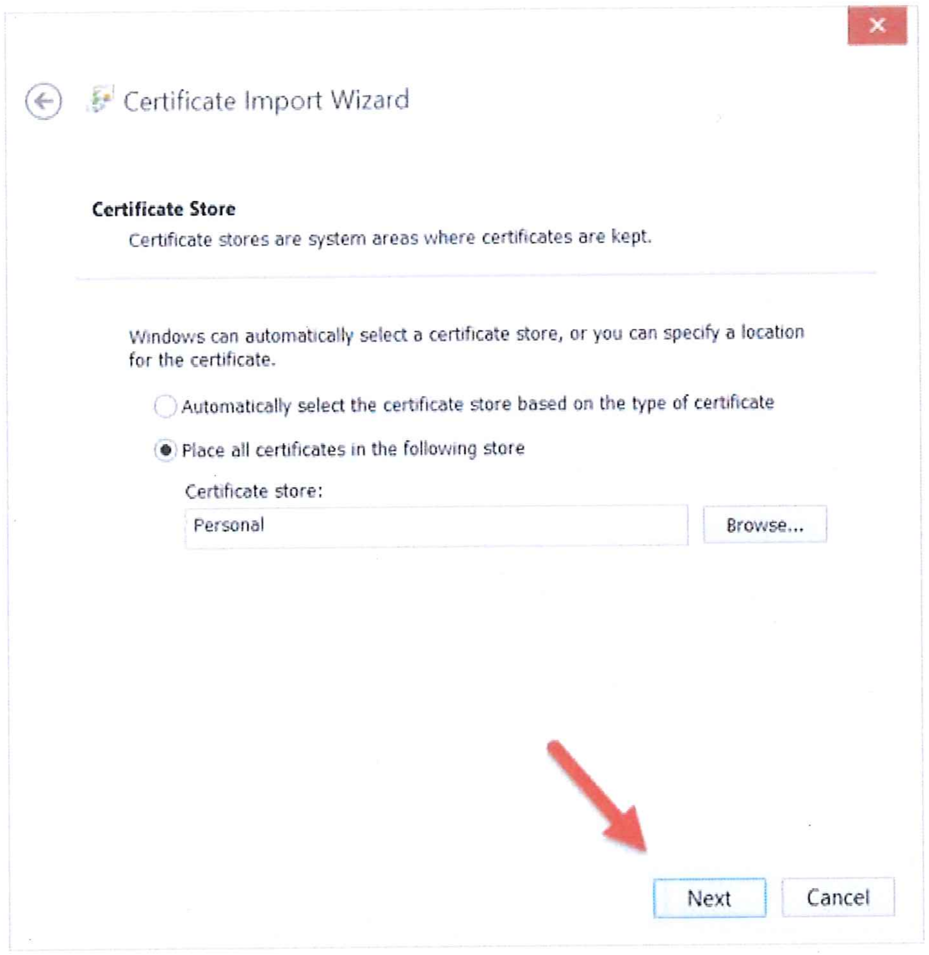
Import options:

☐ Enable strong private key protection. You will be prompted every time the private key is used by an application if you enable this option.

☐ Mark this key as exportable. This will allow you to back up or transport your keys at a later time.

✓ Include all extended properties.

On this window enter the password you associated with the exported cert file and then click on the "Next" button. When you do this the following window will be displayed:


Certificate Import Wizard

Certificate Store

Certificate stores are system areas where certificates are kept.

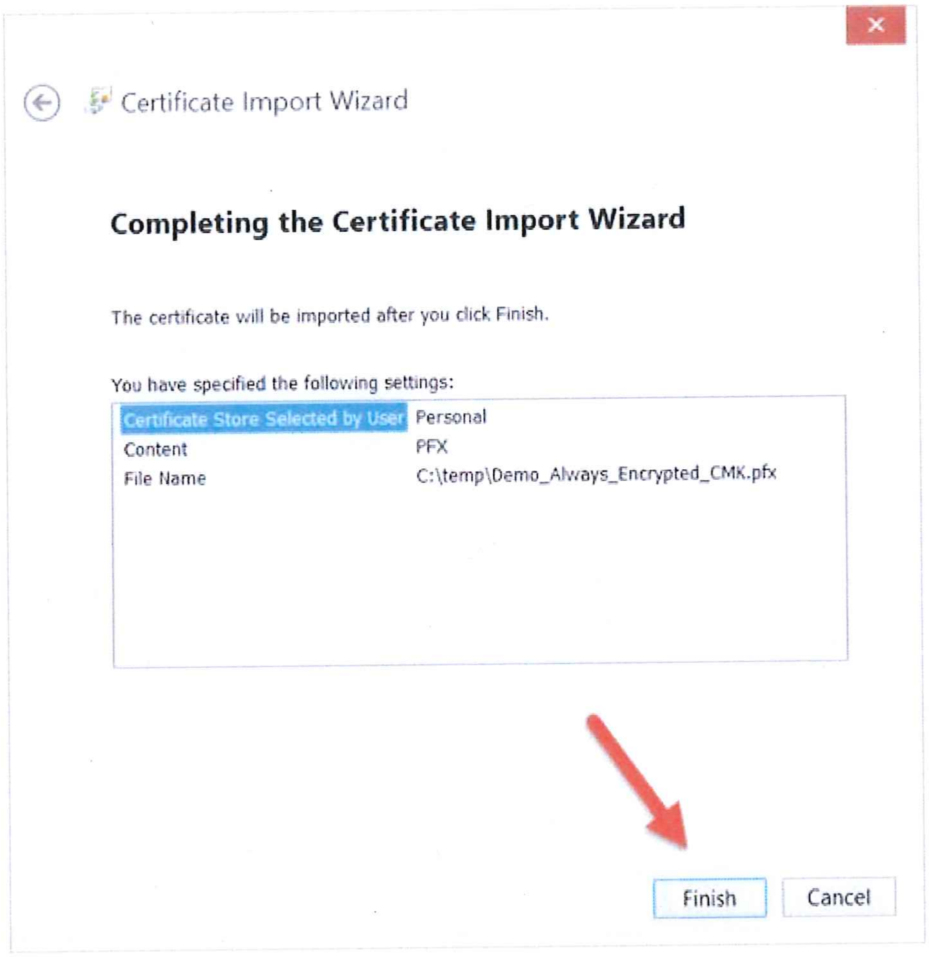
Windows can automatically select a certificate store, or you can specify a location for the certificate.

○ Automatically select the certificate store based on the type of certificate

● Place all certificates in the following store

Certificate store: Personal

On this screen, review where the imported cert was going to be placed. In this case it defaulted to “Personal”. Click on the “Next” button, and the following screen will be displayed:



Certificate Import Wizard

Completing the Certificate Import Wizard

The certificate will be imported after you click Finish.

You have specified the following settings:

Certificate Store Selected by User	Personal
Content	PFX
File Name	C:\temp\Demo_Always_Encrypted_CMK.pfx

On this screen it shows where the imported certificate will be stored and which file was used to import the certificate. Once verifying this information is correct, click on the “Finish” button. When you do this a message box will be displayed verifying successful import of the certificate.

Generating SQL Server Login for C# Application

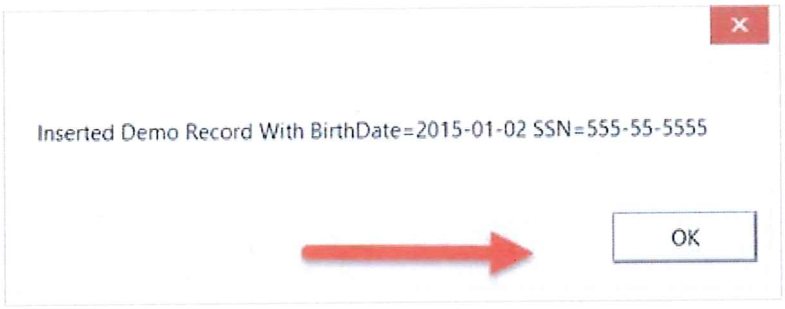
Lastly we need to create a SQL Server login and database user that our C# application will use. Below is the code used to create the SQL authenticated login and database user.

```
--Create SQL Authenticated login
USE [master]
GO
CREATE LOGIN [Test] WITH PASSWORD=N'Test', DEFAULT_DATABASE=[Demo],
CHECK_EXPIRATION=OFF, CHECK_POLICY=OFF
GO

--Create Database User
USE [DEMO]
GO
CREATE USER [Test] FOR LOGIN [Test] WITH DEFAULT_SCHEMA=[dbo]
GO
ALTER ROLE [db_owner] ADD MEMBER [Test]
GO
```

Running Iteration #2 Inserting Always Encrypted Data

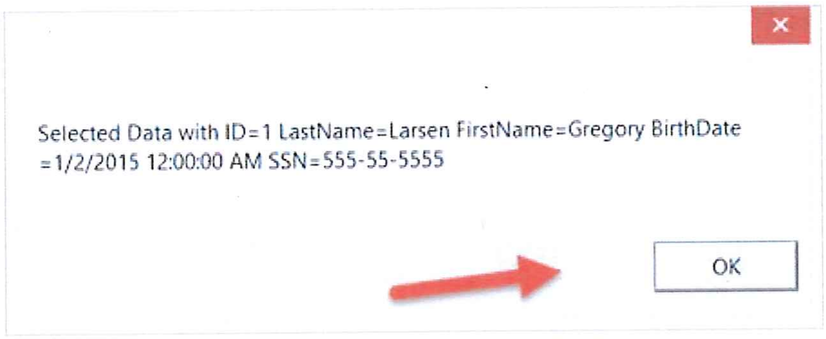
Now that the CMK certificate is stored in the certificate store on the laptop where we will run the C# code that is at the end of this article, in a Visual Studio 2015 .NET 4.6 project. Remember .NET 4.6 framework is the mechanism from encrypting and decrypting Always Encrypted columns. When you execute the C# code via Visual Studio you first see the following Message Box.



```
Inserted Demo Record With BirthDate=2015-01-02 SSN=555-55-5555
```

The text in this message box identifies the values for the BirthDate and SSN columns involved in the INSERT statement. The C# code uses the stored procedure name "Insert_Always_Encrypted" to perform the actual INSERT statement.

When you click "OK" on the message box above the following message box is displayed:



```
Selected Data with ID=1 LastName=Larsen FirstName=Gregory BirthDate
=1/2/2015 12:00:00 AM SSN=555-55-5555
```

This message box displays the information that the C# code read when it runs through the code in the SelectData method. In that method, a simple SELECT statement is used to select data for each column in the Demo_Always_Encrypted table. Before this message box can be displayed, ADO.NET has to decrypt the data in SQL Server since both the BirthDate, and SSN are Always Encrypted columns. As you can see the message box displayed the BirthDate and SSN in clear text. In the next section we will verify the BirthDate and SSN columns are actually encrypted inside of SQL Server table Demo_Always_Encrypted.

Verifying Iteration #2 Inserted Encrypted Data

In order to verify that ADO.NET encrypted the data that we send to the insert stored procedure, we will run the following SELECT statement in a SSMS query window:

```
SELECT [ID]
      ,[LastName]
      ,[FirstName]
      ,[BirthDate]
      ,[SSN]
  FROM [DEMO].[dbo].[Demo_Always_Encrypted];
```

When you run this code you will get the following results:

```
ID      LastName   FirstName BirthDate                              SSN

--      --------   --------- -------------------------------------- --------------------------------

1       Larsen     Gregory   0x011065959924B9E9DE4AA77626F5CCF08... 0x0130F5B22FAD807B9
0653ED072B14...
```

Here you can see that the BirthDate and SSN column are encrypted.

Complete C# Code used in Article:

```
using System.Data;
using System.Data.SqlClient;
using System.Windows.Forms;

// Demo of using Always Encrypted Columns
class AlwaysEncryptedDemo
{
   SqlConnection conn;
   public AlwaysEncryptedDemo()
   {
      // Instantiate the connection
      conn = new SqlConnection(
       "data source=WIN-B95MVF88UT5;initial catalog=Demo;integrated security = False;
         Column Encryption Setting=Enabled; User ID = Greg; Password = Test;");
   }

   // call methods that demo Always Encrypted
   static void Main()
   {
      AlwaysEncryptedDemo scd = new AlwaysEncryptedDemo();
      scd.Insertdata();
      scd.Selectdata();
   }

   public void Insertdata()
   {
      try
      {
         // Open the connection for Insertion
         conn.Open();

         // Constructed command to execute stored proceudre
         string insertString = @"dbo.Insert_Always_Encrypted";

          // Declare variable tho hold insdert command
         SqlCommand icmd = new SqlCommand(insertString, conn);

          //set command type to stored procedure
         icmd.CommandType = CommandType.StoredProcedure;

          // Set value of LastName
         SqlParameter paramLastName = icmd.CreateParameter();
         paramLastName.ParameterName = @"@LastName";
         paramLastName.DbType = DbType.AnsiStringFixedLength; ;
         paramLastName.Direction = ParameterDirection.Input;
```

```
        paramLastName.Value = "Larsen";
        icmd.Parameters.Add(paramLastName);

        // Set value of LastName
        SqlParameter paramFirstName = icmd.CreateParameter();
        paramFirstName.ParameterName = @"@FirstName";
        paramFirstName.DbType = DbType.AnsiStringFixedLength; ;
        paramFirstName.Direction = ParameterDirection.Input;
        paramFirstName.Value = "Greg";
        icmd.Parameters.Add(paramFirstName);

         // Set value of Birth Date
        SqlParameter
        paramBirthdate = icmd.CreateParameter();
        paramBirthdate.ParameterName = @"@BirthDate";
        paramBirthdate.SqlDbType = SqlDbType.Date;
        paramBirthdate.Direction = ParameterDirection.Input;
        paramBirthdate.Value = "2015-01-02";
        icmd.Parameters.Add(paramBirthdate);

        // Set value of SSN
        SqlParameter
        paramSSN = icmd.CreateParameter();
        paramSSN.ParameterName = @"@SSN";
        paramSSN.DbType = DbType.AnsiStringFixedLength;
        paramSSN.Direction = ParameterDirection.Input;
        paramSSN.Value = "555-55-5555";
        paramSSN.Size = 10;
        icmd.Parameters.Add(paramSSN);

        // Exexute Insert
        icmd.ExecuteNonQuery();
        MessageBox.Show("Inserted Demo Record With BirthDate=" + paramBirthdate.Value +
"SSN=" + paramSSN.Value);

      }
      finally
      {
        // Close the connection
        if (conn != null)
        {
          conn.Close();
        }
      }
    }
    public void Selectdata()
    {
      try
      {
        // Open the connection for Selection
        conn.Open();

         // Read Encrypted data
         string selectString = @"SELECT ID, LastName, FirstName, BirthDate, SSN FROM
[dbo].[Demo_Always_Encrypted] ";
        SqlCommand scmd = new SqlCommand(selectString, conn);
        SqlDataReader dataRead = scmd.ExecuteReader();
        while (dataRead.Read())
        {
          MessageBox.Show("Selected Data with ID=" + dataRead["ID"].ToString() +
                        " LastName=" + dataRead["LastName"] +
                        " FirstName=" + dataRead["FirstName"] +
                        " BirthDate =" + dataRead["BirthDate"].ToString() +
                        " SSN=" + dataRead["SSN"].ToString());
            }
        }
        finally
        {
            // Close the connection
            if (conn != null)
            {
                conn.Close();
            }
        }
    }
}
```

References

https://msdn.microsoft.com/en-us/library/mt163865.aspx

https://msdn.microsoft.com/library/w0x726c2.aspx

https://blogs.msdn.microsoft.com/sqlsecurity/2015/06/04/getting-started-with-always-encrypted/

https://msdn.microsoft.com/en-us/library/mt147923.aspx